(Auto)Vacuum and You

Gabrielle Roth

Me.

PDXPUG

RENEWABLE FUNDING

@gorthx

twitter, gmail, wordpress

"I use Postgres because I don't have to care."

Topics

- Vacuum & autovacuum
- A little bit about ANALYZE
- A little bit about MVCC
- Tools
- Fun stories

My first VACUUM.

A long time ago...

- Data "warehouse" of VPN usage
- Nightly addition and ageout of data
- Web front end, report generation

"Hey, it's kinda slow now."

- Did I write some dumb SQL? (No.)
- The adding/deleting rows was the problem.
- I needed to ANALYZE and VACUUM.

Problem 1: Adding "a bunch" of rows

- Query planner uses statistics about data distribution to make decisions about index usage, joins, etc

- Adding (or deleting) "a bunch" of rows that changes the distribution of your data can cause a sub-optimal plan

- ANALYZE updates these statistics.

stats: pg_class

```
pgbench=# SELECT relname, reltuples
FROM pg_class
WHERE relname = 'pgbench_accounts';
-[ RECORD 1 ]---------------
relname   |pgbench_accounts
reltuples |  100002
```

more stats: pg_stats

```
pgbench=# SELECT tablename, attname,
most_common_vals
FROM pg_stats
WHERE tablename = 'pgbench_tellers';
    tablename     | attname   | most_common_vals
------------------+-----------+------------------
 pgbench_tellers | tid       |
 pgbench_tellers | tbalance | {-20716,-5820}
 pgbench_tellers | filler    |
 pgbench_tellers | bid       |
{1,2,3,4,5,...98,99,100}
```

Problem 2: Deleting "a bunch" of rows

Actually, we should talk about MVCC first.

(Have a cocktail.)

A little MVCC.

- Multi-**Version** Concurrency Control
- Allows multiple people to work in the db without @#$%ing things up
- Accomplished in part via transaction ids (xids)
- Take-home message:
  - data changes result in dead/obsolete rows
  - xid wraparound = bad

Problem 2: Deleting "a bunch" of rows

- They're not gone, *you just can't see them*.
- They take up space. ("Bloat".)
- Indexes point to all versions of a row.
- VACUUM fixes this.
- (UPDATEs and rolled-back INSERTs can cause dead rows, too.)

table stats: pg_stat_user_tables

```
pgbench=# SELECT relname,
 n_tup_ins, n_tup_upd, n_tup_del,
 n_live_tup, n_dead_tup,
 last_vacuum, last_analyze
 FROM pg_stat_user_tables
 WHERE relname = 'pgbench_accounts';
 -[ RECORD 1 ]----+------------------------------
relname          | pgbench_accounts
n_tup_ins        | 100000
n_tup_upd        | 73254
n_tup_del        | 0
n_live_tup       | 100002
n_dead_tup       | 4710
last_vacuum      |
last_analyze     | 2014-02-17 20:06:29.900437-08
```

pg_stat_user_tables (cont)

- n_tup_* = incrementing counters; can be reset only by pg_stat_reset
- n_live_tup = this is a guess :)
- n_dead_tup = reset by a vacuum.
- combine the query on the previous slide with \watch for additional fun

more stats: pgstattuple

- contrib module
  - CREATE EXTENSION pgstattuple;
  - One-stop shopping!

```
pgbench=# SELECT tuple_count, tuple_percent,
  dead_tuple_count, dead_tuple_percent
  FROM pgstattuple('pgbench_accounts');
-[ RECORD 1 ]------+-------
tuple_count        | 100000
tuple_percent      | 91.06
dead_tuple_count   | 1592
dead_tuple_percent | 1.45
```

How do I run it?

VACUUM (the manual kind)

- VACUUM
- VACUUM FULL
- VACUUM FREEZE
- VACUUM ANALYZE (...or just ANALYZE)
- must be table owner or superuser

VACUUM

- Removes dead rows
- Cleans up your indexes
- Updates your xids
- (hint bits)
- SHARE UPDATE EXCLUSIVE lock

VACUUM FULL

- Frees up actual disk space
- ACCESS EXCLUSIVE lock
- ...and it's rewriting the table on disk, so you need double the space.
- don't bother if the table's just going to refill.
- http://rhaas.blogspot.com/2014/03/vacuum-full-doesnt-mean-vacuum-but.html

VACUUM FREEZE

- Sets a special xid value: relFrozenXid
- Prevent xid wraparound
- ACCESS EXCLUSIVE lock
- Recommended after very large loads to tables that will see a lot of OLTP

[VACUUM] ANALYZE

- Updates the planner statistics
- SHARE UPDATE EXCLUSIVE
- ANALYZE is actually its own separate thing you can run by itself!
- ANALYZE temp tables after you create them.

VACUUM VERBOSE

```
pgbench=# vacuum verbose pgbench_branches;
INFO:  vacuuming “public.pgbench_branches”
INFO:  index “pgbench_branches_pkey” now contains 1 row
versions in 2 pages
DETAIL:  0 index row versions were removed.
0 index pages have been deleted, 0 are currently
reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  “pgbench_branches”: found 166 removable, 1
nonremovable row versions in 1 out of 1 pages
DETAIL:  0 dead row versions cannot be removed yet.
There were 203 unused item pointers.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
```

Autovacuum!

All my problems are over!

- Available since 8.1

- A "kinder, gentler" vacuum

My table isn't being vacuumed!

(dramatization)

```
SELECT relname, n_live_tup, n_dead_tup,
last_autovacuum, last_autoanalyze
FROM pg_stat_user_tables
WHERE relname = 'pgbench_accounts';
```

```
-[ RECORD 1 ]----+-----------------
relname          | pgbench_accounts
n_live_tup       | 1000000
n_dead_tup       | 9499
last_autovacuum  |
last_autoanalyze |
```

Is autovacuum even on?

- ps -ef | grep vacuum

```
postgres 1101 972 0 06:37 ? 00:00:33 postgres: autovacuum
launcher
process
```

- in postgresql.conf:

```
autovacuum = on #default
track_counts = true #default
```

- psql shell:

```
pgbench=# SELECT name, setting || unit AS setting FROM
pg_settings
WHERE category = 'Autovacuum'; pgbench=# SHOW autovacuum;
```

- Verify that track_counts is enabled, too

At what point is a vacuum triggered?

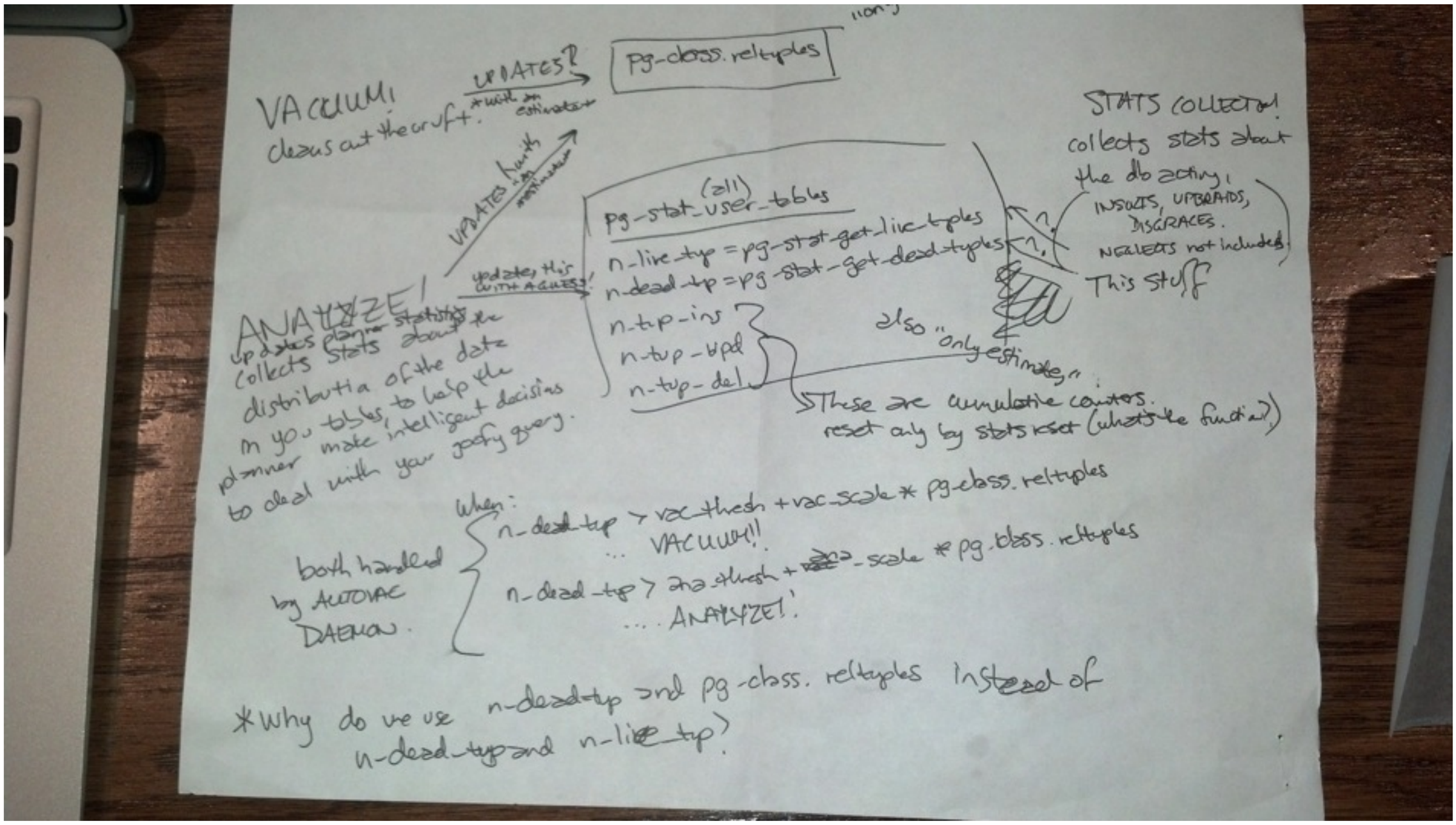
- in postgresql.conf:

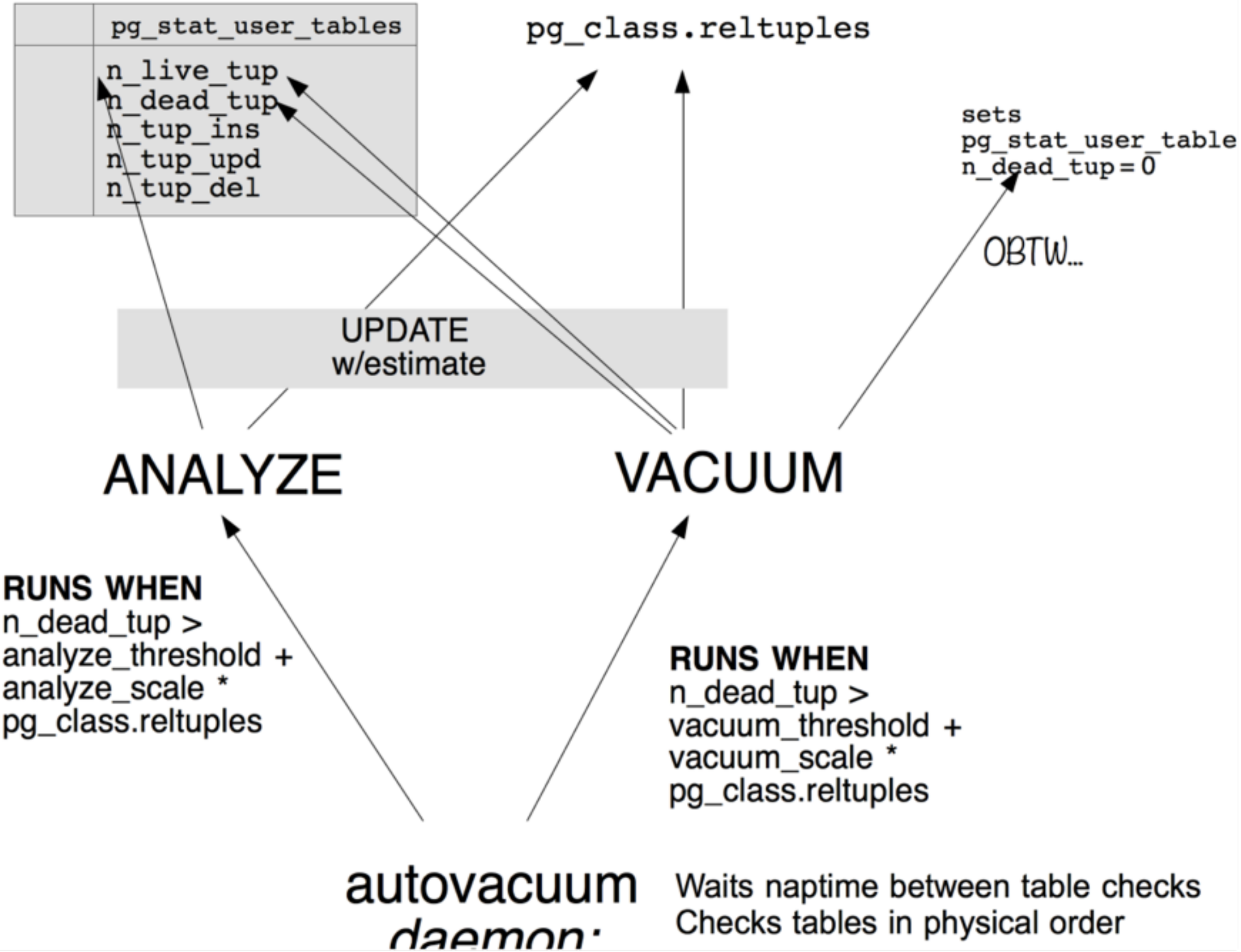
```
#autovacuum_vacuum_threshold = 50
    # min number of row updates before vacuum
#autovacuum_vacuum_scale_factor = 0.2
    # fraction of table size before vacuum
```

autovacuum: do the math.

- vacuum threshold = autovacuum_vacuum_threshold + autovacuum_vacuum_scale_factor * pgclass.reltuples
- 1,000,000 row table = 50 + (0.2 * 1000000) = 200,050 9500 dead tuples is not even close to triggering a vacuum

How this is supposed to work.




pg_stat_user_tables
n_live_tup
n_dead_tup
n_tup_ins
n_tup_upd
n_tup_del
pg_class.reltuples
sets
pg_stat_user_table
n_dead_tup = 0
OBTW...
UPDATE
w/estimate
ANALYZE
VACUUM
RUNS WHEN
n_dead_tup >
analyze_threshold +
analyze_scale *
pg_class.reltuples
RUNS WHEN
n_dead_tup >
vacuum_threshold +
vacuum_scale *
pg_class.reltuples
autovacuum
daemon:
Waits naptime between table checks
Checks tables in physical order

Tuning

GUCs of particular interest

- autovacuum_vacuum_threshold
- autovacuum_vacuum_scale_factor
- autovacuum_max_workers
- autovacuum_nap_time
- autovacuum_cost_limit
- autovacuum_cost_delay

GUCS+

- autovacuum_analyze_threshold and scale_factor
- autovacuum_freeze_max_age
- Note that you will get a vac freeze to prevent wraparound even if you have autovacuum disabled.
- autovacuum_multixact_freeze_max_age (9.3+)
- autovacuum_work_mem (9.4?)

Before we begin...

- Back up your config!
- Have metrics
- Make use of 'include' in postgresql.conf
- log_autovacuum_min_duration = [YMMV]
- Collect table stats (just for kicks)

sample log message

```
log_autovacuum_min_duration = 0

%LOG:  automatic vacuum of table
"ttrss.public.ttrss_feedbrowser_cache": index scans: 1
pages: 0 removed, 11 remain
tuples: 303 removed, 303 remain
buffer usage: 82 hits, 0 misses, 10 dirtied
avg read rate: 0.000 MB/s, avg write rate: 3.585 MB/s
system usage: CPU 0.00s/0.00u sec elapsed 0.02 sec

%LOG:  automatic analyze of table
"ttrss.public.ttrss_feedbrowser_cache" system usage:
CPU
0.00s/0.00u sec elapsed 0.03 sec
```

GUCs: when will vac happen

#autovacuum_vacuum_threshold = 50
    # min number of row updates before vacuum

#autovacuum_vacuum_scale_factor = 0.2

    # fraction of table size before vacuum

live_tup	default	0 rows, 0.2 sf	100k rows, 0 sf
1,000	250	200	100,000
10,000	2,050	2,000	100,000
100,000	20,050	20,000	100,000
1,000,000	200,050	200,000	100,000
10,000,000	2,000,050	2,000,000	100,000
100,000,000	20,000,050	20,000,000	100,000
1,000,000,000	200,000,050	200,000,000	100,000

GUCs: how many tables can be vacced at ~ the same time

- #autovacuum_max_workers = 3
  # max number of autovacuum subprocesses
  - **requires a restart**
- #autovacuum_naptime = 1min
- # time between autovacuum runs
- These are per-cluster.
- As you add workers, they'll go slower.
- Be mindful of maintenance_work_mem if you are on < 9.4: keep av_max_workers * maint_work_mem < memory

GUCs: How fast can I make this thing go

- #autovacuum_vacuum_cost_limit = -1
  # default vacuum cost limit for autovacuum; -1 means use vacuum_cost_limit (default: 200 ”credits”)
- #autovacuum_vacuum_cost_delay = 20ms
  # default vacuum cost delay for autovacuum, in milliseconds; -1 means use vacuum_cost_delay (default: 0ms)
- speed this up by:
  - increasing cost_limit to some value in the hundreds, or (and?)
  - setting cost_delay to 0

Caveats!

- All of these GUCs that we just looked at* interact together.
- Dramatic changes in table size may require adjustments
- You still need to manually:
    - VACUUM [FREEZE] ANALYZE after large data loads
    - ANALYZE temp tables
- Isn't this fun?

*and some others outside the scope of this talk

per-table adjustment

- can't do this with naptime or max_workers
- `CREATE TABLE mytable (blahblah) WITH (autovacuum_vacuum_threshold = 2000);`
- `ALTER TABLE mytable SET (autovacuum_vacuum_threshold = 5000);`
- view with \d+:

```
Options: autovacuum_vacuum_threshold=5000
```

- -- reset to value from postgresql.conf!

```
ALTER TABLE mytable RESET
(autovacuum_vacuum_threshold);
```

Other fun things I've encountered

OH !#@*&(%!!!
(reenactment)

```
        relname  | ins | upd | del | live | dead | l_aa | l_av
-----------------+-----+-----+-----+------+------+------+------
pgbench_branches |   0 |   0 |   0 |    0 |    0 |      |
pgbench_tellers  |   0 |   0 |   0 |    0 |    0 |      |
pgbench_history  |   0 |   0 |   0 |    0 |    0 |      |
pgbench_accounts |   0 |   0 |   0 |    0 |    0 |      |
```

streaming rep + vacuum

- table stats don't get replicated
- (planner stats do, but we can't see those)
- You can't run VACUUM on a standby:

```
postgres=# vacuum mytable;
ERROR:  cannot execute VACUUM during recovery
```

- vacuum jobs are WAL logged

Orphan temp tables

```
LOG:  autovacuum: found orphan temp table
"pg_temp_5444"."feeds" in database "ttrss"
```

Skipped tables

```
2014-09-12 01:44:25.583
   PDT,,,30540,,4dbffb0c.7b5b,5,,2014-09-12
   01:41:42 PDT,74/868,0,LOG,55P03,"skipping
   analyze of ""foo"" --- lock not
available",,,,,,
```

Inheritance

- VACUUM/ANALYZE on individual tables only
- per-table config settings aren't inherited either

Wishlist

- An easier way to see what's being vacuumed & the progress thereof
  - Can use a combo of ps & looking at pg_locks hoping to catch something going by
- A way to view the vacuum queue & see WHO'S NEXT.

Help! (and further reading)

- Pg docs + -admin + Pg wiki https://wiki.postgresql.org/wiki/VacuumHeadaches
- xid wraparound: https://devcenter.heroku.com/articles/postgresql-concurrency
- Josh B's ”Freezing Your Tuples Off” series
- http://rhaas.blogspot.com/2011/03/troubleshooting-stuck-vacuums.html
- http://rhaas.blogspot.com/2014/03/vacuum-full-doesnt-mean-vacuum-but.html

Thank you!

PgConf.EU

SPI

PDXPUG

Selena Deckelmann

Robert Haas

Simon Riggs